The University of Alberta

THE COMPLEXITY OF A DELIVERY PROBLEM

by

©

P. Lisa Higham

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Master of Science

Department of Computing Science

Edmonton, Alberta
Fall, 1982

ABSTRACT

The complexity of a one-dimensional, fixed capacity delivery problem is examined. An instance of the problem is specified by n pairs of points (trips) on a straight line segment and a capacity k. Each pair of points represents a source and destination of a package and the capacity is the limit on the allowable load of the delivery car. The car is required to travel back and forth on the line segment picking up packages at sources and delivering them to their corresponding destinations, in such a way that the car's capacity never exceeds k. The delivery car may not unload a package temporarily. Therefore, once a source is visited the package remains part of the car's cargo until its destination is reached. The focus here is on a further restriction, that each trip has the same direction in the line segment. It is required to find a route for the delivery car that completes all deliveries in a minimum distance.

A polynomial time algorithm is presented for the capacity $= 1$ problem. The capacity ≥ 3 problem is shown to be *NP*-complete.

ACKNOWLEDGEMENTS

I would like to thank the members of my supervisory committee Len Schubert, Ted Lewis and Jeff Pelletier. The attention and time they gave to this thesis is very much appreciated. Thanks are especially due to my supervisor Francis Chin, who introduced me to this area of research and directed me toward a solution. Jeff Sampson and Carol Smith persevered with a rough draft of this thesis, and contributed improvements of both content and style. I am grateful to Steve Sutphen for encouragement and help while discovering the wonders of UNIX and its text formatting system.

Above all, my deepest thanks to Alan, for his unending patience and kindness in addition to his generous assistance.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The boundary between those problems that can be solved in polynomial time and those that likely have no polynomial solution has been of interest to computing scientists since Cook's paper [1972]. This thesis presents a problem called **the delivery problem** that has a polynomial solution in its most restricted version. However, by relaxing only one parameter, it joins the class of problems for which the only known algorithms require a number of steps that grows exponentially with the size of the problem. It therefore becomes apparently intractable.

1.1. Informal Description of the Delivery Problem

The most general form of the delivery problem can be thought of as one of optimizing the path of a delivery car which must haul a number of packages from given sources to given destinations without temporary unloading. The car has a finite capacity and each package uses up a specific amount of this capacity. As will be seen, this problem is too general to be interesting. Various restrictions can be placed on the problem and will be discussed in Chapter 2. However the focus of this thesis is on the following very restricted version:

> n pairs of points on a straight line segment are given, each representing a source and destination of a package. A delivery car is required to run back and forth on the line segment, picking up and delivering these packages with the restriction that it can carry no more than k (k a fixed parameter greater than or equal to one) packages at one time. The car may not unload a package temporarily. Therefore, once a source is visited, the package remains part of the car's cargo until its destination is reached. A further restriction is that all deliveries point in the same direction. That is, if (s,d) is a source-destination pair, then $d>s$. It is required to find a route for the car which completes all deliveries in the minimum distance.

Note that this is a significantly more restricted problem than the general one because:

1. The car travels in only one dimension.

2. All packages take up an equal capacity, 1, of the car's capacity, k.

3. All deliveries point in the same direction.[1]

Note also that the car may start anywhere and end anywhere so long as it travels a minimum distance. This restricted problem is called the **one way, one-dimensional, k capacity delivery problem**. It is abbreviated as ***1Way del-k***.

A summary of necessary definitions and concepts follows. A much more thorough treatment of the following material can be found in any source dealing with complexity of algorithms. This summary is loosely based on Garey and Johnson [1979].

1.2. Time Complexity of Algorithms

The time complexity of an algorithm is a statement of its time requirements as a function of the input size. Suppose a problem, R, is solved by an algorithm, A, with (time) complexity f. This means that for every integer n, A solves every instance of R with input length n, in time no greater than $f(n)$. Clearly the time required depends on both the measure and the computer model used. Similarly input length is dependent on the encoding scheme for the algorithm. The resulting time complexity functions, however, vary by no more than a polynomial function. These distinctions are immaterial in the theory of *NP*-completeness, which is concerned with identifying broad categories of algorithm complexity.

A function $f:I \rightarrow I$ is ***O(g)*** if and only if there exists a constant c such that $|f(n)| \leq c\,|g(n)|$ for all n greater than some finite positive integer N.

A **polynomial time algorithm** is one whose time complexity function f is $O(p)$ for some polynomial function p. Thus there exists a constant c such that for every integer n, $|f(n)| \leq c\,|p(n)|$ for all input of length n.

[1]This thesis will denote this direction as forward and represent it in figures as upward.

A problem for which no polynomial time algorithm exists is considered **intractable.**

1.3. Decision and Optimization Problems

Frequently two different versions of a problem can be considered, one requiring an optimization, the other requiring a decision. This is the case for the delivery problem of this thesis. The two versions are:

Optimization problem

Instance: n source-destination pairs and a capacity k.

Problem: Find a feasible route with least length that completes all deliveries.

Decision problem

Instance: n source-destination pairs, a capacity k, and a bound B.

Question: Does there exist a feasible route of length less than or equal to B, that completes all deliveries?

A solution is **feasible** if it meets the restrictions of the problem. In this case a route is feasible if the car never carries more than k packages at a time.

The set of all instances of a decision problem, R, contains as a subset those instances which result in a yes decision. This subset is the set of **yes-instances** denoted by $\boldsymbol{R_Y}$.

In order to be precise, the theory of NP-completeness is applied to decision problems only. However any algorithm that solves the optimization version of a problem, can be converted to one that solves the corresponding decision problem by just appending a comparison of the result with the bound B. Frequently the converse is also true. Using the decision algorithm, a binary search can be done on the possible outcomes to determine the optimal bound B^*. If the set of possible outcomes is small enough, this search takes no more than polynomial time. Next the optimal solution is constructed sequentially from partial solutions in polynomial time. This is done with a step by step extension of intermediate results until the final solution is found.

1.4. Polynomial Reducibility

A decision problem R is **reducible** to a decision problem Q if there exists a constructive transformation which converts any instance I of R into a corresponding instance I' of Q such that I is a yes-instance of R if and only if I' is a yes-instance of Q.

If the transformation can be constructed in time bounded by a polynomial function of the input size, then R is **polynomially reducible** to Q.

1.5. NP-completeness and Intractability

There exist many provably intractable problems. However so far, those that have been proven intractable are either undecidable or intractable even using the non-deterministic model for computation.[2] There is another class of problems suspected of being intractable, but no proof yet exists. Finally, there are the provably tractable problems. These last two classes are distinguished as follows.

NP is the class of all decision problems which can be solved in polynomial time using a non-deterministic model of computation.[3]

P is the class of all problems that can be solved in polynomial time using a deterministic model of computation.

Since any algorithm for a deterministic Turing machine can be converted to one for a non-deterministic Turing machine by ignoring the extra power of the non-deterministic model, the class P is a subset of the class NP.

A fundamental open problem of computer science today is whether or not P is a proper subset of NP. Although it is widely conjectured that $P \neq NP$, there is yet no problem known to be in NP and yet intractable.

[2]Definitions and discussion of deterministic and non-deterministic models of computation can be found in Garey and Johnson [1979].

[3]Though these definitions and that of *NP*-completeness to follow are properly stated in the form of language recognition problems on deterministic and non-deterministic Turing machines, for the purposes of this thesis the equivalent characterization in terms of algorithms is sufficient.

Instead, following the example of Cook, attention is focused on those problems that are among the *hardest* problems in *NP*. Cook proved [1971] that every problem in *NP* can be polynomially reduced to one particular problem in *NP* called **Satisfiability.** Subsequently Karp [1972], and then many others, proved that a great number of decision problems shared this property with satisfiability. The importance of this class of problems has been recognized and given a name.

A problem is ***NP*-complete** if it is in *NP* and every problem in *NP* can be polynomially reduced to it.

Proving that a new problem is *NP*-complete is simplified by Cook's pioneering work. Since polynomial reduction is transitive, it is sufficient to show two things. A problem, *R*, is *NP*-complete if:

1. *R* is in *NP*. That is, there is a polynomial time non-deterministic algorithm that solves *R*.

2. There is an *NP*-complete problem that is polynomially reducible to *R*.

1.6. Preview

In Chapter 2 the general delivery problem is discussed and some variations are shown to be trivially *NP*-complete. Other related problems are also cited. In Chapter 3 a polynomial algorithm for the one way, one dimensional, capacity one delivery problem, ($1Way\ del{-}1$), is developed and analyzed. For $k \geq 3$, the one way, one dimensional, capacity *k*, delivery problem, ($1Way\ del{-}k$), is *NP*-complete. Proof of this is the content of Chapter 4. Chapter 5 contains a summary of results, some conclusions and observations, and describes some related open problems.

CHAPTER 2

RELATED PROBLEMS

2.1. The General Delivery Problem

The **general delivery problem** does not restrict the dimension of the space in which the trips lie. Nor is the weight of each package or the total car capacity restricted, though they are assumed to be positive integers.[1] The definition of the decision problem version of the general delivery problem is:

Decision general delivery

Instance: n pairs of points (s_i, d_i) in Euclidean m-space, an n-tuple $W=(w_1, \cdots, w_n)$ associating a weight w_i with each pair (s_i, d_i), a capacity k, and a bound B.

Question: Is there a route for the car of length no more than B, such that it completes all deliveries and the total weight in the car at any one time is $\leq k$?

The *1Way del-k* problem is a subproblem of the general delivery problem. Other subproblems may be generated by changing the restrictions on the general problem.

In light of the large degree of generality, it is not surprising that the general delivery problem is *NP*-complete. In fact, the problem remains *NP*-complete when one or two of three different parameters are significantly restricted. Consider the following possible restrictions:

[1]Real numbers are not considered since it is intended to show that these problems are inherently *NP*-complete, not just difficult because of the complexity of computing with real numbers. No loss of generality results from eliminating rational numbers since a problem with rational weights and capacity can be converted to an equivalent integer problem by multiplying by the greatest common denominator.

parameter	unrestricted value	restricted value
dimension	positive integer m	1
weights	w_i=*arbitrary integer* $\leq k$ for all i	$w_i=1$ for all i
capacity	positive integer k	1

Except when the array of weights is arbitrary and the capacity restricted to 1, all combinations of these restrictions have meaning. However, the problem remains *NP*-complete unless all three restrictions are imposed simultaneously.

The notation ***delivery(mD,wW,kcap)*** will be used here to represent the delivery problem with parameters m (dimension), w (weight), and k (capacity) replaced from the table above. For example, *delivery*$(1D,arbW,kcap)$ is the one dimensional, arbitrary weights, k capacity delivery problem. Using this notation, the *1Way del-k* problem becomes *delivery*$(1D,1W,kcap)$ with the additional restriction that all deliveries are in the same direction.

2.2. Some NP-complete Subproblems

Some *NP*-complete results are immediate. A non-deterministic algorithm can solve the general delivery problem in polynomial time. After guessing a solution, it is only necessary to check that the proposed route is feasible, and to add up the length of the route to ensure it is less than or equal to B. Since both these last two operations can be done in linear time, the general delivery problem is in *NP*, and hence all subproblems are as well. Therefore only the second requirement for *NP*-completeness, a constructive reduction, will be sketched.

2.2.1. The Delivery(2D,1W,1cap) Problem

Theorem 2.1

Delivery$(2D,1W,1cap)$ is *NP*-complete.

proof:

The reduction is from the **Euclidean Travelling Salesman** *(ETS)* problem for the plane. This problem is *NP*-complete [Papadimitriou,1977]. Let an instance, I, of *ETS* be specified by a

collection, C, of n cities, a complete set, E, of intercity distances defined by the discrete Euclidean metric,[2] and a bound B. The transformation to an instance of *delivery*$(2D,1W,1cap)$ consists of constructing a very short trip, (s_i,d_i), starting at each city c_i in C. The idea is that, if the trips are short enough, they will have negligible effect on the length of a route either between the cities in ETS or through the deliveries in *delivery*$(2D,1W,1cap)$.

Construction of an instance, I' of *delivery*$(2D,1W,1cap)$ corresponding to I in ETS is as follows. At each city c_i in ETS construct a source s_i in *delivery*$(2D,1W,1cap)$. In order to ensure integer values throughout, the edges in E and the bound B are enlarged by multiplying the distance function by $3n$ and the new trips are given unit length. Thus $\delta(s_i,s_j) = 3n\,\delta(c_i,c_j)$.[3] At each s_i construct a trip (s_i,d_i) of length one. Finally, set $B' = 3nB + 2n$. This construction can be performed in polynomial time and has the property that I is a yes-instance of ETS if and only if I' is a yes-instance of *delivery*$(2D,1W,1cap)$.

For suppose there is a route $(c_{\alpha_1}, \cdots, c_{\alpha_n})$ of length $\leq B$ for I. Then the path $(s_{\alpha_1}, \cdots, s_{\alpha_n})$ has length $\leq 3nB$. Hence, by the triangle inequality of the distance function, the route $(s_{\alpha_1}, d_{\alpha_1}, \cdots, s_{\alpha_n}, d_{\alpha_n})$ has a length $\leq 3nB + 2n$ for I'. Thus $I \in ETS_Y$ implies $I' \in$ *delivery*$(2D,1W,1cap)_Y$. Conversely, suppose I' has a route of length $\leq B' = 3nB + 2n$. Then because of the scale factor, I has a route of length $\leq \frac{3nB+2n}{3n} = B + \frac{2}{3}$. But all edges have integer length. Therefore I has a route of length $\leq B$. Thus $I' \in$ *delivery*$(2D,1W,1cap)_Y$ implies $I \in ETS_Y$. □

2.2.2. The Delivery(1D,arbW,kcap) Problem

Theorem 2.2

Delivery$(1D,arbW,kcap)$ is *NP*-complete.

[2] This metric is defined for the plane by $\left\lceil \left((x_1-x_2)^2 + (y_1-y_2)^2\right)^{\frac{1}{2}} \right\rceil$ where $\lceil x \rceil$ is defined as the smallest integer greater than or equal to x.

[3] $\delta(x,y)$ is the distance from x to y.

proof:

The reduction is from the *NP*-complete problem **Partition** [Karp,1972]. Let I be an instance of Partition specified by a set, $A = \{a_1, \ldots, a_n\}$ of n positive integers.[4] The corresponding instance, I' of *delivery*$(1D,arbW,kcap)$ contains n trips T_i=(1,2) of unit length, each with the same source and destination where the weight of trip T_i is $2a_i$. Capacity k is set to $\sum_{i=1}^{n} a_i$ and the bound to 3. Then there is a route of length 3 if and only if all deliveries can be made in two forward passes. This is possible if and only if there is a partition of A into 2 equal parts. □

The fact that *delivery*$(1D,1W,kcap)$ is also *NP*-complete is a consequence of a slightly more restricted problem, (the one way version), proved *NP*-complete in Chapter 4.

2.3. The Polynomial Subproblem

When the specified restrictions are placed on all three parameters, the *delivery*$(1D,1W,1cap)$ problem results. A car travels back and forth on a straight line segment, making the trips from s_i to d_i one at a time. The car is allowed to start anywhere and end anywhere. The problem is to find a shortest route.

Because the car's capacity is one, after picking up a package at its source, it must travel directly to the corresponding destination. Therefore, for a specific instance, every route contains the same trips. Only the extra distance between trips can vary. Hence, minimizing this overhead is equivalent to minimizing the route length.

Let A be an n by n array, where $A_{i,j}$ is the distance from the i^{th} destination to the j^{th} source, and $A_{i,i}=\infty$. The problem, then, is the same as finding a single cycle permutation $(i_1=i_{n+1}, \ldots, i_n)$ of $(1, \ldots, n)$ such that $\left[\sum_{j=1}^{n} A_{i_j,i_{j+1}} - \max_{1\le j\le n} \{A_{i_j,i_{j+1}}\}\right]$ is minimal.[5]

[4] The problem Partition asks if there is a subset A' of A such that $\sum_{a_i \in A} a_i = \sum_{a_i \in A-A'} a_i$.

[5] Finding a single cycle permutation $(i_1=i_{n+1}, \ldots, i_n)$ of $(1, \ldots, n)$ such that $\left[\sum_{j=1}^{n} A_{i_j,i_{j+1}}\right]$ is minimal yields a minimum tour rather than a minimum route.

This formulation highlights the similarity to the Travelling Salesman (route) problem. But contrary to the the claim in an unpublished manuscript by Chang [1976], the problems are not equivalent. The distance array for the Travelling Salesman problem may have arbitrary entries. In *delivery*(1*D*,1*W*,1*cap*), the matrix entries are more restricted than Chang assumed. The restriction to one dimension implies that the rows of matrix A are not independent. Thus *delivery*(1*D*,1*W*,1*cap*) is a subproblem of the Travelling Salesman problem.

Gilmore and Gomory [1964] solved a closely related problem. Their $O(n \log n)$ algorithm for sequencing a one state-variable machine, when applied to the input of *delivery*(1*D*,1*W*,1*cap*), produces the shortest *tour* for the delivery car, rather than the shortest *route*. If a fixed starting point, b, and an endpoint, e, are specified, their algorithm could still be used by including the trip (e,b) in the set of trips, and finding the shortest tour. But there are only n^2-n possible start and end point combinations in *delivery*(1*D*,1*W*,1*cap*) that could yield optimal routes. Therefore Gilmore's algorithm could be applied n^2-n times to yield the optimal route with arbitrary start and end points. Though this solution takes $O(n^3 \log n)$ and appears inefficient, it proves that *delivery*(1*D*,1*W*,1*cap*) is indeed in P.

An approach quite different from Gilmore and Gomory's yields an $O(n \log n)$ solution provided that, in addition to the current restrictions, all trips have the same direction. This algorithm is described in the next chapter. It is likely that a better algorithm than the $O(n^3 \log n)$ method mentioned above is possible for the two directional case as well.

2.4. Other Related Problems

Common features shared by various forms of the delivery problem and each of the Travelling Salesman, the Euclidean Travelling Salesman, and Partition problems have already been illustrated. Many other problems bear some degree of similarity to the delivery problem.

An elevator problem discussed in Knuth's Volume Three [1973], and solved by Karp, considers the optimal transport of people between floors, using a single elevator. This problem resembles *delivery*(1*D*,1*W*,*kcap*) but has additional restrictions that admit a polynomial time solution

unlike *delivery*$(1D, 1W, kcap)$. Though these extra properties result in quite a different problem, the technique used to calculate a lower bound resembles that used in Chapter 4 to facilitate proof of *NP*-completeness of the k capacity delivery problem.

Wong, Liu, and Apter [1973] proved that a very simple algorithm transfers records on a rotating drum in optimal or nearly optimal time. Their problem is related to *1Way del-1*. However the drum scheduling problem operates on a circle instead of a line segment. In addition, the drum rotates constantly in one direction, whereas in *1Way del-1* the delivery car can move in both directions. These crucial differences mean that their solution does not carry over to the delivery problem.

A number of scheduling and sequencing problems have some superficial resemblance to various forms of the delivery problem. But those problems fail to capture the position information inherent in the sources and destinations of *1Way del-k*. Also, they often include other restraints such as a partial ordering or multiple processors that are not analogous to parameters of the delivery problem.

CHAPTER 3

THE ONE WAY, CAPACITY ONE DELIVERY PROBLEM

In this chapter a polynomial solution for the one directional delivery problem where capacity equals one is developed. The complexity of the algorithm is analyzed and is shown to be of optimal order.

3.1. Initial Definitions and Terminology

In order to make the problem and its solution precise the following definitions are used.

A **trip,** T is an ordered pair (s,d) of integers. The functions ***src***(T) and ***dest***(T) select the first and second coordinates of T respectively. Let $\Gamma = \{T_1, \ldots, T_n\}$ be a collection of trips. The function ***ls*** selects the trip $T_j \in \Gamma$ such that $src(T_j) \leq src(T_i)$ for every $T_i \in \Gamma$. Thus, $ls(\Gamma)$ selects the trip in Γ with the least source. Similarly, ***gd***(Γ) selects the trip with the greatest destination. That is, $gd(\Gamma) = T_l$ if and only if $dest(T_l) \geq dest(T_i)$ for every i. The expression ***top***(Γ) is an abbreviation for $dest(gd(\Gamma))$. Similarly, ***bottom***(Γ) is an abbreviation for $src(ls(\Gamma))$. The interval spanned by the trips in Γ, $\overline{bottom(\Gamma)\ top(\Gamma)}$, is denoted ***I***$(\Gamma)$. A set of trips Γ is **gapless** if, for any $p \in I(\Gamma)$ there exists an i such that $src(T_i) \leq p \leq dest(T_i)$. Let Γ contain n trips and α be an arrangement of the set $\{1, \ldots, n\}$. A **route** R_α is an ordering of the n trips in Γ, according to α. Thus $R_\alpha = (T_{\alpha_1} T_{\alpha_2}, \ldots, T_{\alpha_n})$. The **length,** ***Len,*** of a route R_α is: $Len(R_\alpha) = \sum_{i=1}^{n} | d_{\alpha_i} - s_{\alpha_i} | + \sum_{i=1}^{n-1} | d_{\alpha_i} - s_{\alpha_{i+1}} |$. The **overhead,** ***Ov***(R_α) of a route, R_α is: $Ov(R_\alpha) = \sum_{i=1}^{n-1} | d_{\alpha_i} - s_{\alpha_{i+1}} |$. The pairs $(d_{\alpha_i}, s_{\alpha_{i+1}})$ are the **links** of the route R_α. Links are **backward links** if $d_{\alpha_i} \geq s_{\alpha_{i+1}}$ and **forward links** otherwise. A **subroute** of Γ is an arrangement of a subset of the trips in Γ.

3.2. Problem Definition

1Way del-1 is defined by:

Instance: a set Γ of n trips in a line segment $\overline{PQ}$ such that $s_i \leq d_i$ for every $T_i \in \Gamma$.

Problem: Find the arrangement γ such that

$$Len(R_\gamma) = minimum\{Len(R_\alpha) \mid \alpha \text{ is an arrangement of } 1, \ldots, n\}.$$

For the remainder of this chapter, all trips are in one direction (forward).

3.3. Results

For all arrangements α, the first term of $Len(R_\alpha)$, is $\sum_{i=1}^{n} | d_{\alpha_i} - s_{\alpha_i} | = \sum_{i=1}^{n}(d_i - s_i)$. This is just the sum of the lengths of the trips in Γ and is constant for all routes. Therefore, to minimize the route length it is sufficient to find the route with minimum overhead. This is equivalent to minimizing the sum of the lengths of the links.

3.3.1. Lower Bound on Overhead

Let R_α be any route through Γ. Then:

$$\begin{aligned} Ov(R_\alpha) &= \sum_{i=1}^{n-1} | d_{\alpha_i} - s_{\alpha_{i+1}} | \\ &\geq \sum_{i=1}^{n-1} d_{\alpha_i} - \sum_{i=1}^{n-1} s_{\alpha_{i+1}} \\ &= \sum_{i=1}^{n} d_i - d_{\alpha_n} - \sum_{i=1}^{n} s_i + s_{\alpha_1} \\ &\geq \sum_{i=1}^{n} d_i - \sum_{i=1}^{n} s_i - top(\Gamma) + bottom(\Gamma) \end{aligned}$$

Denote this lower bound by $\boldsymbol{LB}(\Gamma)$. Denote an optimal route through Γ by R^*. Then $LB(\Gamma)$ is a lower bound for $Ov(R^*)$ and any route R that has $Ov(R) = LB(\Gamma)$ must be an optimal route.

1Way del-1 requires that the minimum length be found over the set of all possible start and end points. However a similar lower bound is useful for a route with fixed start and end points. Let R'_α be a route with fixed start and end points b and e respectively. Then:

$$
\begin{aligned}
Ov(R'_\alpha) &= |b - s_{\alpha_1}| + \sum_{i=1}^{n-1} |d_{\alpha_i} - s_{\alpha_{i+1}}| + |d_{\alpha_n} - e| \\
&\geq \sum_{i=1}^{n} d_i - \sum_{i=1}^{n} s_i + b - e.
\end{aligned}
$$

Denote this lower bound by ***FLB***(Γ).

Notice that $LB(\Gamma)$ is achieved by a route R_γ if and only if $T_{\gamma_1} = ls(\Gamma)$, $T_{\gamma_n} = gd(\Gamma)$, and $d_{\gamma_i} \geq s_{\gamma_{i+1}}$ for every i. That is, a route achieves the lower bound $LB(\Gamma)$, if and only if it begins at the least source, ends at the greatest destination, and all links are backward links. Similarly, a route achieves lower bound $FLB(\Gamma)$ if and only if it travels backward from b to s_{γ_1}, from d_{γ_n} to e, and between trips.

3.3.2. Staircases

A **staircase** for Γ is a subroute $ST = (St_1, \ldots, St_m)$ of trips in Γ such that $St_1 = ls(\Gamma)$, $St_m = gd(\Gamma)$, and $src(St_{i+1}) \leq dest(St_i) \leq dest(St_{i+1})$ for every $i \leq m-1$.

Lemma L3.1

If Γ is gapless, then a staircase for Γ can be constructed in time $O(|\Gamma|)$.

proof:

Let $\Gamma = \{T_1, \ldots, T_n\}$ be ordered so that $src(T_i) \leq src(T_{i+1})$. A staircase for this gapless set can be constructed as follows.

```
$St_1 \leftarrow T_1$                    # $T_1 = ls(\Gamma)$ #
$j \leftarrow 1$; $i \leftarrow 2$
while $St_j \neq gd(\Gamma)$
do
        while $dest(T_i) < dest(St_j)$
        do
                $i \leftarrow i+1$
        od
        $j \leftarrow j+1$
        $St_j \leftarrow T_i$
        $i \leftarrow i+1$
od.
```

Notice that in the inner loop there must exist an i such that $dest(T_i) \geq dest(St_j)$ since $top(\Gamma) \geq dest(St_j)$. Also the first trip, T, such that $dest(T) \geq dest(St_j)$ must have

$src(T) \leq dest(St_j)$ since otherwise Γ would not be gapless. Since i is incremented at every pass through the inner loop, the algorithm examines each trip in Γ once and therefore requires time $O(|\Gamma|)$. □

Lemma L3.2

Let Γ be a set of gapless trips together with a fixed start b and end e such that $b, e \in I(\Gamma)$. If $b \geq e$, then there is a route R' through Γ such that $Ov(R')$ achieves the lower bound $FLB(\Gamma)$, and thus is optimal for fixed start and end.

proof:

Let $ST = \{St_1, \ldots, St_m\} \subset \Gamma$ be a staircase for Γ. Let H be those trips $T \in (\Gamma - ST)$ such that $src(T) \geq b$. Similarly, let L be the set of trips $T' \in (\Gamma - ST)$ such that $src(T') < b$. Then $\Gamma = (ST \bigcup H \bigcup L)$. Order H and L from greatest destination to least destination yielding $(H_1, \ldots, H_h)$ and $(L_1, \ldots, L_l)$. Now let $R' = (b, L_1, \ldots, L_l, St_1, \ldots, St_m, H_1, \ldots, H_h, e)$. Since $b \geq e$, R' has only backward links and therefore achieves lower bound $FLB(\Gamma)$. □

This lemma yields an upper bound on the overhead of the optimal route R^* through a gapless set Γ. Let p be some point, $p \in I(\Gamma)$. By lemma L3.2 there is a route R_α which starts and ends at p and achieves $FLB(\Gamma)$. Therefore,

$$\begin{aligned} Ov(R_\alpha) &= FLB(\Gamma) \\ &= \sum_{i=1}^{n} d_i - \sum_{i=1}^{n} s_i - p + p \\ &= \sum_{i=1}^{n} d_i - \sum_{i=1}^{n} s_i \end{aligned}$$

Denote this upper bound for a gapless set Γ by $\boldsymbol{UB}(\Gamma)$.

3.3.3. Additional Length of a Route

For any route R through Γ, $Ov(R)$ can be calculated directly from the forward links in R and its start and end points. This is the content of the following lemma.

Lemma L3.3

Let R_α be a route through Γ which starts at $src(T_{\alpha_1}) = s_{\alpha_1}$ and ends at $dest(T_{\alpha_n}) = d_{\alpha_n}$.

Then

$$Ov(R_\alpha) = LB(\Gamma) + 2\sum_{fl}(s_{\alpha_{i+1}} - d_{\alpha_i}) + (s_{\alpha_1} - bottom(\Gamma)) + (top(\Gamma) - s_{\alpha_n})$$

where *fl* is the set of forward links.

proof:

Let *bl* and *fl* represent the set of backward links and forward links respectively.

$$\begin{aligned}
Ov(R_\alpha) &= \sum_{i=1}^{n-1} | d_{\alpha_i} - s_{\alpha_{i+1}} | \\
&= \sum_{bl}(d_{\alpha_i} - s_{\alpha_{i+1}}) + \sum_{fl}(s_{\alpha_{i+1}} - d_{\alpha_i}) \\
&= \sum_{bl}(d_{\alpha_i} - s_{\alpha_{i+1}}) + \sum_{fl}(d_{\alpha_i} - s_{\alpha_{i+1}}) + 2\sum_{fl}(s_{\alpha_{i+1}} - d_{\alpha_i}) \\
&= \sum_{i=1}^{n-1}(d_{\alpha_i} - s_{\alpha_{i+1}}) + 2\sum_{fl}(s_{\alpha_{i+1}} - d_{\alpha_i}) \\
&= \sum_{i=1}^{n} d_i - d_{\alpha_n} - \sum_{i=1}^{n} s_i + s_{\alpha_1} + 2\sum_{fl}(s_{\alpha_{i+1}} - d_{\alpha_i}) \\
&= LB(\Gamma) + 2\sum_{fl}(s_{\alpha_{i+1}} - d_{\alpha_i}) + (s_{\alpha_1} - bottom(\Gamma)) + (top(\Gamma) - d_{\alpha_n}).\square
\end{aligned}$$

Thus $Ov(R_\alpha)$ is greater than $LB(\Gamma)$ by twice the sum of its forward links plus the difference between its start and end points and the points $bottom(\Gamma)$ and $top(\Gamma)$ respectively.

3.3.4. Improved Bounds

If Γ is a set of trips with gaps, then the lower bound $LB(\Gamma)$ can be improved. The following lemmas show that an optimal route through Γ has forward links through the gaps in a set of trips.

Lemma L3.4

Suppose $\Gamma = \Gamma_1 \bigcup \Gamma_2 \bigcup \cdots \Gamma_t$ where Γ_i and Γ_{i+1} are sets of gapless trips arranged from lowest to highest sources and $I(\Gamma_i) \bigcap I(\Gamma_{i+1}) = \emptyset$. Let the gap between Γ_i and Γ_{i+1} be called G_i and have size g_i. That is, the distance from $top(\Gamma_i)$ to $bottom(\Gamma_{i+1})$ is g_i. Then if R is a route through Γ, $Ov(R) \geq LB(\Gamma) + 2\sum_{i=1}^{t-1} g_i$.

proof:

Let the start of R be b and the end be e.[1] For every h such that G_h lies above b, R must travel forward across G_h, since it must complete all trips. Similarly, for every l such that G_l lies below e, R must travel forward across G_l. Now consider the gaps that are both below b and above e. If there are no such gaps, then the result follows by lemma L3.3. Otherwise, $(b - bottom(\Gamma)) > \sum_{G_i < b} g_i$ and $(top(\Gamma) - e) > \sum_{e < G_i} g_i$. Therefore, using L3.3

$$\begin{aligned} Ov(R) &\geq LB(\Gamma) + 2\sum_{G_i > b} g_i + 2\sum_{G_i < e} g_i + (b - bottom(\Gamma)) + (top(\Gamma) - e) \\ &\geq LB(\Gamma) + 2\sum_{i=1}^{t-1} g_i . \square \end{aligned}$$

This improved lower bound for the overhead of a route through the trips in $\Gamma = \Gamma_1, \ldots, \Gamma_t$ with gaps $G_1, \ldots, G_{t-1}$, is called ***ILB***(Γ).

An upper bound for a route through a set of trips Γ, possibly with gaps, can now be formulated. Let $\Gamma = \Gamma_1 \bigcup \cdots \bigcup \Gamma_t$ be as in lemma L3.4, and let ST_i be a staircase for Γ_i. Let R_α be a route that starts at $bottom(\Gamma)$, completes each staircase, ST_i, in order from bottom to top, and then completes the trips in $\Gamma_i - ST_i$ from top to bottom in order of decreasing sources finally ending in Γ_1. The only forward links for this route are across the gaps G_i. So by lemma L3.3

$$\begin{aligned} Ov(R) &= LB(\Gamma) + 2\sum_{fl}(s_{\alpha_{i+1}} - d_{\alpha_i}) + (top(\Gamma) - bottom(\Gamma)) \\ &= LB(\Gamma) + 2\sum_{i=1}^{t-1} g_i + (top(\Gamma) - bottom(\Gamma)) \\ &= \sum_{i=1}^{n} d_i - \sum_{i=1}^{n} s_i + 2\sum_{i=1}^{t-1} g_i . \end{aligned}$$

Therefore the optimal route must have overhead less than or equal to this length. This length is denoted by ***IUB***(Γ).

Lemma L3.5

For any set of one-directional trips Γ, there is a route, R^*, of optimal length which starts at a point lower than or equal to its ending point.

[1] It is sufficient to assume $b = src(T_i)$ and $e = dest(T_j)$ for some i and j, since otherwise R can be made shorter by eliminating this initial and final travel.

proof:

Let R_α be a route through Γ with start b greater than end e. Assume $\Gamma = \Gamma_1 \bigcup \cdots \bigcup \Gamma_t$ as in lemma L3.4.

Case 1: Suppose b and e are in the same interval $I(\Gamma_l)$. Then by lemma L3.3, $Ov(R_\alpha) = LB(\Gamma) + 2\sum_{fl}(s_{\alpha_{i+1}} - d_{\alpha_i}) + (top(\Gamma) - e) + (b - bottom(\Gamma))$. But all gaps, G_i above b or below e must have forward links through them. Therefore, in this case, all gaps have forward links. Thus

$$Ov(R) \geq LB(\Gamma) + 2\sum_{i=1}^{t-1} g_i + (top(\Gamma) - e) + (b - bottom(\Gamma)).$$

Since $b > e$ this is greater than $IUB(\Gamma)$. Therefore R cannot be optimal.

Case 2: Suppose $b \in I(\Gamma_k)$ and $e \in I(\Gamma_j)$ where $k > j$. By lemma L3.3

$$Ov(R_\alpha) = LB(\Gamma) + 2\sum_{fl}(s_{\alpha_{i+1}} - d_{\alpha_i}) + (top(\Gamma) - e) + (b - bottom(\Gamma)).$$

All gaps above b, and all gaps below e must have forward links. Therefore

$$\begin{aligned} Ov(R_\alpha) &\geq LB(\Gamma) + 2\sum_{i \geq k} g_i + 2\sum_{i<j} g_i + (top(\Gamma) - e) + (b - bottom(\Gamma)) \\ &= LB(\Gamma) + 2\sum_{i \geq k} g_i + 2\sum_{i<j} g_i + 2(b - e) + (top(\Gamma) - b) + (e - bottom(\Gamma)) \end{aligned}$$

But $2(b - e) = 2\sum_{i=j}^{k-1} g_i + 2\sum_{j<i<k} I(\Gamma_i) + 2(b - bottom(\Gamma_k)) + 2(top(\Gamma_j) - e)$

Therefore

$$\begin{aligned} Ov(R_\alpha) \geq LB(\Gamma) &+ 2\sum_{i=1}^{t-1} g_i + 2\sum_{j<i<k} I(\Gamma_i) + 2(b - bottom(\Gamma_k)) + 2(top(\Gamma_j) - e) \\ &+ (top(\Gamma) - b) + (e - bottom(\Gamma)). \end{aligned}$$

But a trip, R' can be constructed that begins at $top(\Gamma_j)$ and ends at $bottom(\Gamma_k)$ and has overhead no greater than $Ov(R_\alpha)$. First construct a staircase, ST_i for each set Γ_i. The route R' proceeds as follows:

1. for i from j down to 1, do the trips in $(\Gamma - ST_i)$ in order of decreasing sources.

2. for i from 1 to j do the trips in ST_i in order.

3. for i from $j+1$ to $k-1$ do the trips in Γ_i by completing ST_i in order, and then completing $(\Gamma_i - ST_i)$ in order of decreasing sources.

4. for i from k to t do the trips in ST_i in order.

5. for i from t down to k do the trips in $(\Gamma_i - ST_i)$ in order of decreasing sources.

In steps 1 and 5, there are no forward links. In steps 2 and 4 the only forward links are across gaps. In step 3, for each i, there is a forward link from the destination of the final trip in $(\Gamma_i - ST_i)$ to $top(\Gamma_i)$ which continues across gap G_i. This forward length must be less than $g_i + I(\Gamma_i)$. Therefore, using lemma L3.3, the overhead of R' can be bounded above.

$$Ov(R') \leq LB(\Gamma) + 2\sum_{i=1}^{t-1} g_i + 2\sum_{j<i<k} I(\Gamma_i) + (top(\Gamma) - bottom(\Gamma_k))$$
$$+ (top(\Gamma_j) - bottom(\Gamma)).$$

This is less than or equal to the overhead of route R_α. Therefore R' is at least as short a route as R_α. □

3.3.5. Additional Overhead of Subroutes

Suppose $S_\alpha = (S_{\alpha_1}, \ldots, S_{\alpha_m})$ is a subroute of Γ containing exactly the trips in $\Gamma' \subset \Gamma$. It is useful to determine how good the ordering of S_α is assuming that these are the only trips to be covered. The **additional overhead**, ***AOv,*** of a subroute S_α measures the amount of overhead accumulated by S_α in excess of the lower bound for the set of trips in S_α. It is defined by $AOv(S_\alpha) = Ov(S_\alpha) - ILB(\Gamma')$. Thus $AOv(S_\alpha) = 0$ if and only if S_α starts at the $ls(\Gamma')$, ends at $gd(\Gamma')$ and has backward links except across gaps. Otherwise

$AOv(S_\alpha) = 2\sum_{fl}(s_{\alpha_{l+1}} - d_{\alpha_l}) - 2\sum_{G_i \in \Gamma'} g_i + (src(S_{\alpha_1}) - bottom(\Gamma')) + (top(\Gamma') - dest(S_{\alpha_m}))$. For a complete route R_α, $AOv(R_\alpha) = 0$ implies that R_α achieves the lower bound and therefore must be optimal.

3.4. The Solution to 1Way del-1

The optimal route for a set of trips $\Gamma = \Gamma_1 \bigcup \cdots \bigcup \Gamma_t$ is found by determining a route, R^* such that $AOv(R^*)$ is minimal. This proceeds in three steps. First, a subroute, S^* is constructed such that 1) $AOv(S^*) = 0$, 2) S^* contains $ls(\Gamma_i)$ and $gd(\Gamma_i)$ for all i, and 3) S^* is maximal in the sense that there is no subroute S' containing a proper superset of the trips in S^* that also has $AOv(S') = 0$. The second step is to insert the remaining trips into S^* in such a way that minimum additional overhead is accumulated. The final step is to prove that the route thus constructed is optimal.

3.4.1. First Part of the Solution

Let $\Gamma = \Gamma_1 \bigcup, \ldots, \bigcup \Gamma_t$ where each Γ_i is a maximal gapless subset. Assume $|\Gamma_i| = m(i)$, that Γ is arranged from lowest to highest set, and that the trips in each subset $\Gamma_i = \{T_{i_1}, \ldots, T_{i_{m(i)}}\}$ are ordered by increasing $src(T_{i_j})$. By lemma L3.1 there exists a staircase $ST_i = (St_{i_1}, \ldots, St_{i_{s(i)}})$ for each set of trips Γ_i. These staircases can be joined together from lowest to highest to yield a subroute S of Γ,

$S = (St_{1_1}, \ldots, St_{1_{s(1)}}, St_{2_1}, \ldots, St_{2_{s(2)}}, \ldots, St_{t_1}, \ldots, St_{t_{s(t)}})$. The additional overhead of S is zero.

Let W be the set $\Gamma - \bigcup_{i=1}^{t} ST_i$ of trips not in any staircase, and let $W_1, \ldots, W_w$ be the maximum gapless subsets of W arranged from lowest to highest. Some of these W_i can be inserted into S yielding a longer subroute of Γ in such a way that the additional overhead remains zero.

Lemma L3.6

Let $S = (S_1, \ldots, S_p)$ be the subroute built from concatenating staircases together, and denote the interval, $\overline{src(S_{j+1})\ dest(S_j)}$, spanned by the link from S_j to S_{j+1} by $I(link_j)$. If there exists a j such that $I(W_i) \bigcap I(link_j) \neq \emptyset$ then W_i can be inserted into S without increasing the additional overhead.

proof:

Suppose $I(W_i) \bigcap I(link_j) \neq \emptyset$. Because of the construction of the staircases, the intersection must be with a backward link, and therefore $dest(S_j) \geq src(S_{j+1})$. But W_i is gapless. So by lemma L3.2, there is a route R_w through W_i starting at $b = sup(I(W_i) \bigcap I(link_j))$ and ending at $e = inf(I(W_i) \bigcap I(link_j))$ with only backward links. Let S' be the route that results from inserting R_w between S_j and S_{j+1}. Since S' still starts at $ls(\Gamma)$ and ends at $gd(\Gamma)$ and has only backward links except across gaps, the additional overhead remains 0. □

Let S^* be the subroute built by inserting into S all such W_i that have non-empty intersection with some stairway link as in L3.6. If there exists W_i up to W_{i+r} all intersecting the same stairway link, $link_j$, then each of them can be inserted between St_j and St_{j+1} in order from highest to lowest set. Because of its construction, S^*, clearly satisfies the first two of the three properties listed at the beginning of this section, that is: 1) $AOv(S^*) = 0$ and 2) S^* contains $ls(\Gamma_i)$ and $gd(\Gamma_i)$ for every i.

3.4.2. Second Part of the Solution

If every set W_i has been inserted into S^* then S^* is a route for Γ with $AOv(S^*) = 0$. It therefore has overhead that achieves $ILB(\Gamma)$ and hence is an optimal route.

In general however, there may remain some subsets which do not intersect a staircase link. Rename these uninserted sets from lowest to highest $V_1, \ldots, V_v$, and denote the collection $\{V_1, \ldots, V_v\}$ by Λ. For each of these sets the interval, $I(V_k)$, is contained in the interval $\overline{s_j\ d_j}$ for exactly one trip S_j in S^*, and $I(V_k)$ intersects no other trips outside of V_k. Figure 3.1 illustrates the type of configuration possible for some of these remaining sets which are represented by ellipses.

It is important to observe that the sets in Λ are determined by the structure of Γ and are independent of the staircase construction. A different staircase yields exactly the same set of trips as those in S^*, and the same uninserted sets $V_1, \ldots, V_v$. Also, as the following lemma shows, S^* satisfies the third property, namely 3) S^* is a maximal subroute for which the properties 1)

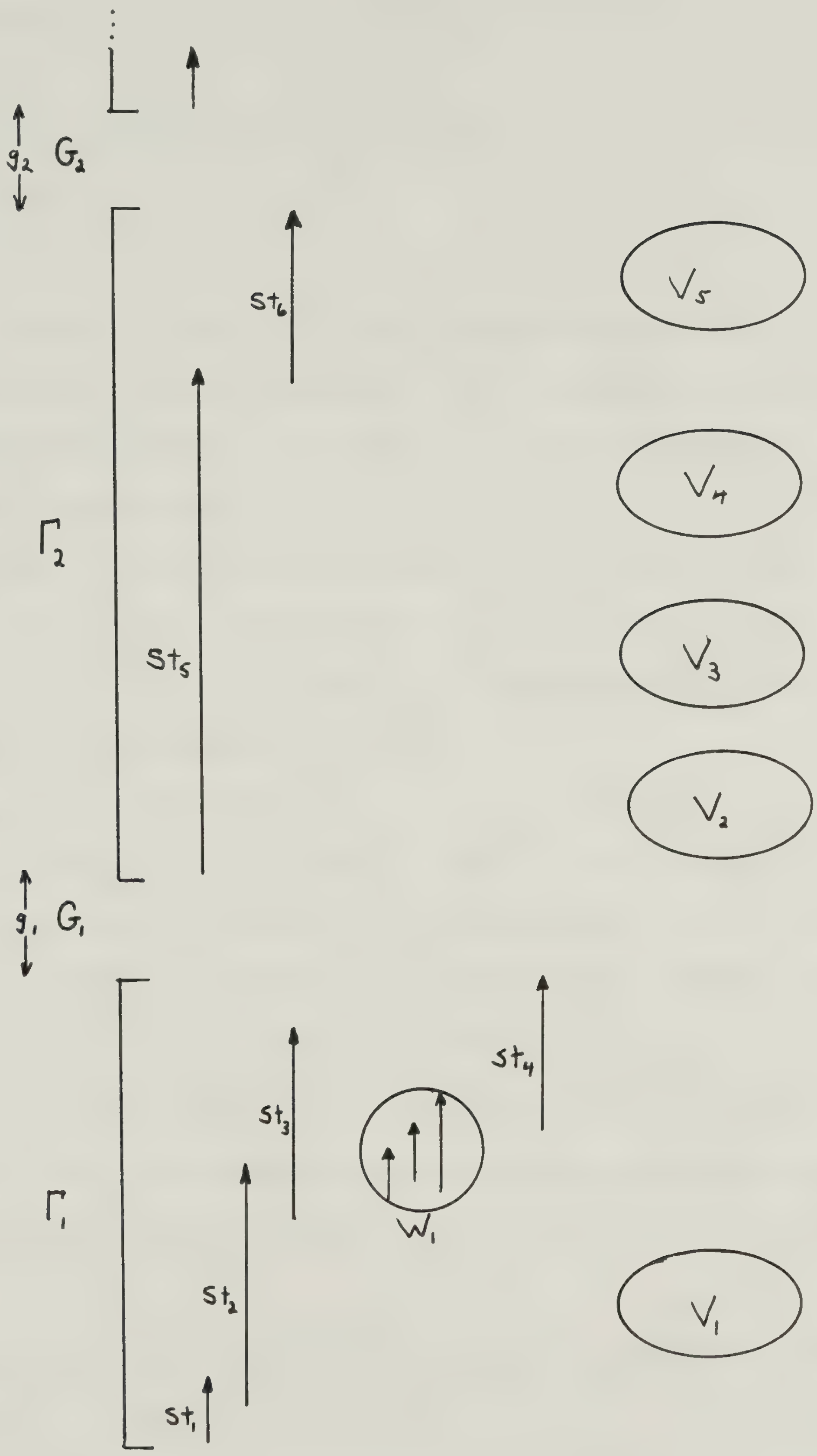


Figure 3.1 Sample configuration of some V-sets

and 2) above hold.

Lemma L3.7

S^* is a maximal subroute for Γ such that 1) $AOv(S^*) = 0$ and 2) S^* contains $ls(\Gamma_i)$ and $gd(\Gamma_i)$ for every i.

proof:

Let T be any trip in $\Gamma - S^*$ and let $S_j \in S^*$ be the trip such that $I(T) \subset I(S_j)$. Let V_T be the V-set in Λ containing T. Consider any subroute, S' of Γ that includes both S_j and T and also includes $ls(\Gamma_i)$ and $gd(\Gamma_i)$ for every i. Then either S_j precedes T or T precedes S_j.

Case I: Suppose S_j precedes T in S'.

Let T' be the first trip after T in S' such that $src(T') > dest(T)$. If any such T' exists, then S' must contain a forward link at least as great as the distance from $dest(T)$ to $\min\{src(T'), dest(S_j)\}$. If no such T' exists, then S' must terminate at some point less than or equal to $top(V_T)$. In either situation, $AOv(S') > 0$.

Case II: Suppose S_j succeeds T in S'.

Let T'' be the last trip before T in S' such that $dest(T'') < src(T)$. If any such T'' exists, then S' must contain a forward link at least as great as the distance from $\max\{dest(T''), src(S_j)\}$ to $src(T)$. If no such T'' exists, then S' must begin at some point greater than or equal to $bottom(V_T)$. Again, in either situation, $AOv(S') > 0$.

Thus no trip can be added to those in S^* while maintaining an additional overhead of 0. □

This part of the solution inserts the trips in Λ in such a way that they add a minimal amount of additional overhead. Each one of the sets in Λ is added to the subroute S^* in one of three ways.

1. V_k can be prepended to S^*. That is no trips lower than $bottom(V_k)$ are completed before V_k is begun.

2. V_k can be appended to S^*. That is no trips higher than $top(V_k)$ are started after V_k is completed.

3. V_k can be inserted into another part of the subroute. That is either trips with sources lower than $bottom(V_k)$ occur both before and after V_k, or trips with destinations higher than $top(V_k)$ occur both before and after V_k.

Methods 1. and 2. change the start and end points respectively but can be done without adding forward links. Method 3. does not change the start and end points of the subroute but adds forward links. Therefore any of the options add additional overhead to the new subroute. It is necessary to determine which of these three options is optimal for each V_k, and construct a final route R^* from this information. It must also be shown that any other route through Γ would accumulate at least as much additional overhead as R^*.

The following lemma presents a way to prepend trips to S^* (corresponding to method 1.) without need for any forward links, and calculates the additional overhead due to this construction.

Lemma L3.8

The sets $V_1, \dots, V_i$ can be prepended to S^* yielding $<i>S^*$ in such a way that $AOv(<i>S^*) = bottom(V_i) - bottom(\Gamma)$.

proof:

Let $ST(V_i)$ be a staircase for V_i. Let the subroute $<i>S^*$ be defined by prepending the trips in sets $V_1, \dots, V_i$ to S^*. $<i>S^*$ begins at $bottom(V_i)$. It first completes the trips in $ST(V_i)$, then those in $V_i - ST(V_i)$ in order of decreasing sources. Next for k from $i-1$ to 1 it completes the set V_k in order of decreasing sources. Finally S^* is appended *as is* to this initial subroute. There are no new forward links in $<i>S^*$ so $AOv(<i>S^*) = bottom(V_i) - bottom(\Gamma)$. □

The trips in sets $V_l, \dots, V_v$ can be appended to S^* in a way analogous to the prepending of lemma L3.8. Thus after completing S^*, the sets V_v down to V_{l+1} are appended in order of decreasing sources. Finally the set V_l is appended by completing $V_l - ST(V_l)$ and then climbing the staircase $ST(V_l)$. The following lemma results.

Lemma L3.9

The sets $V_l, \ldots, V_v$ can be appended to S^* yielding $S^*<l>$ in such a way that $AOv(S^*<l>) = top(\Gamma) - top(V_l)$.

Let $<i>S^*<l>$ be the subroute with the trips in $V_1, \ldots, V_i$ prepended to S^* and $V_l, \ldots, V_v$ appended to S^* as in lemmas L3.8 and L3.9. All trips except those in $\bigcup_{k=i+1}^{l-1} V_k$ are included in this subroute. These remaining trips can be inserted into $<i>S^*<l>$ and the additional overhead due to these insertions can be determined.

It is convenient to rename each of the remaining sets V_k, $i+1 \leq k \leq l-1$ in a way that reflects which staircase trip, S_j, contains $I(V_k)$. Suppose $I(S_j)$ contains $I(V_q), \ldots, I(V_r)$, $i+1 \leq q \leq r \leq l-1$ but not $I(V_{q-1})$ nor $I(V_{r+1})$. Then alternatively label $V_q, \ldots, V_r$ as $V'_{j_1}, \ldots, V'_{j_{\mu(j)}}$ where $\mu(j) = r-q+1$ indicating that $I(V'_{j_t}) \subset \overline{s_j\, d_j}$ for $1 \leq t \leq \mu(j)$.

Some further definitions are necessary. Refer to Figure 3.2 which illustrates some of the sets of trips embedded in one staircase trip. Let θ_{j_k} be the greatest destination less than $bottom(V'_{j_k})$. Note that θ_{j_k} is $top(V'_{j_{k-1}})$ if $k>1$ and θ_{j_1} is the destination of some element in $<i>S^*<l>$. If there is no destination less than $bottom(V'_{j_k})$, (only the case when $i=0$, $j=1$, $k=1$), then set $\theta_{j_k} = bottom(\Gamma)$. Similarly let σ_{j_k} be the least source greater than $top(V'_{j_k})$. Then σ_{j_k} is $bottom(V'_{j_{k+1}})$ if $k<\mu(j)$ and $\sigma_{j_{\mu(j)}}$ is the source of some trip in $<i>S^*<l>$. If there is no source greater than $top(V'_{j_k})$, (only possible when $l=v+1$, $V'_{j_k}=V_v$ and is contained in $gd(\Gamma)$), then set $\sigma_{j_k} = top(\Gamma)$. Let δ_{j_k} be the distance from $top(V'_{j_k})$ to $bottom(V'_{j_{k+1}})$ for $1 \leq k \leq \mu(j)-1$. Also let δ_{j_0} be the distance from $bottom(V'_{j_1})$ to θ_{j_1} and let $\delta_{j_{\mu(j)}}$ be the distance from $top(V'_{j_{\mu(j)}})$ to $\sigma_{j_{\mu(j)}}$.

The following lemma inserts $V_{i+1}, \ldots, V_{l-1}$ into $<i>S^*<l>$ using exactly $\mu(j)$ forward links for each collection of sets $V'_{j_1}, \ldots, V'_{j_{\mu(j)}}$. It uses the smallest available set of forward

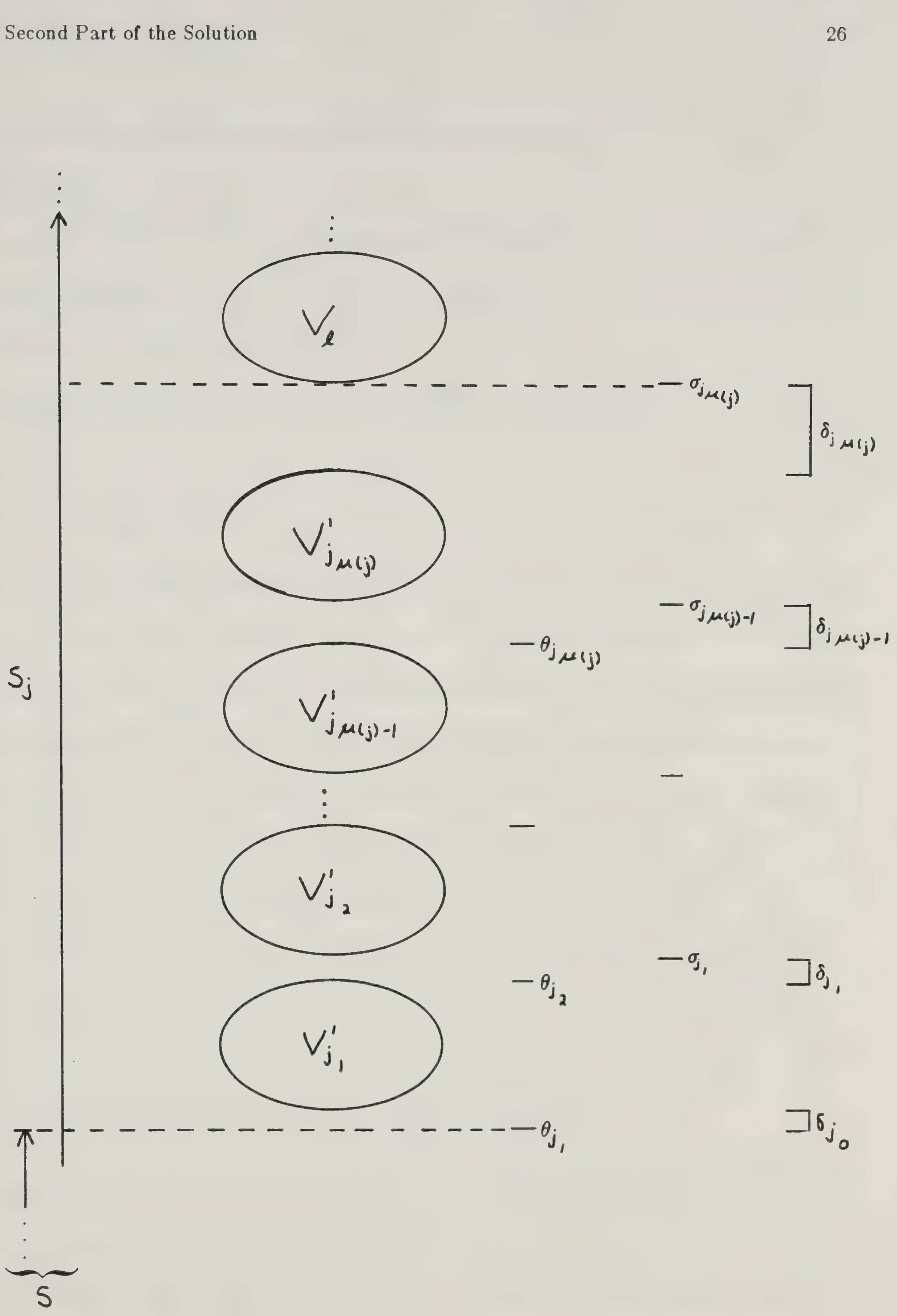


Figure 3.2 Notation used for V-sets

links, and determines the additional overhead due to this construction.

Lemma L3.10

Suppose $I(V'_{j_1}), \ldots, I(V'_{j_{\mu(j)}})$ are contained in S_j and let θ_{j_k}, σ_{j_k}, and δ_{j_k} be defined as above. Then the sets $V'_{j_1}, \ldots, V'_{j_{\mu(j)}}$ can be inserted into $<i>S^*<l>$ yielding a longer subroute $<i>S_j^*<l>$ in such a way that

$$AOv(<i>S_j^*<l>) = AOv(<i>S^*<l>) + 2\sum_{k=0}^{\mu(j)} \delta_{j_k} - 2 \max_{0\leq k\leq \mu(j)} \{\delta_{j_k}\}.$$

proof:

Let $m(j)$ be the index such that $\max_{0\leq k\leq \mu(j)} \{\delta_{j_k}\} = \delta_{j_{m(j)}}$. and let ST_{j_k} be a staircase for V'_{j_k} for $1\leq k\leq \mu(j)$. The extended subroute $<i>S_j^*<l>$ has the trips in V'_{j_1} up to $V'_{j_{m(j)}}$ inserted into $<i>S^*<l>$ after θ_{j_1}, and has $V'_{j_{m(j)+1}}$ up to $V'_{j_{\mu(j)}}$ inserted into $<i>S^*<l>$ before $\sigma_{j_{\mu(j)}}$. Form a subroute $S_{j,1}$ that first travels up the staircases $ST_{j_1}, \ldots, ST_{j_{m(j)}}$ and then completes the trips in $V'_{j_{m(j)}} - ST_{j_{m(j)}}$ down to $V'_{j_1} - ST_{j_1}$ in order of decreasing sources. Also construct the subroute $S_{j,2}$ that first completes the trips in $V'_{j_{\mu(j)}} - ST_{j_{\mu(j)}}$ down to $V'_{j_{m(j)+1}} - ST_{j_{m(j)+1}}$ in order of decreasing sources, and then travels up the staircases from $ST_{j_{m(j)+1}}$ to $ST_{j_{\mu(j)}}$. Then the subroute $<i>S_j^*<l>$ is the subroute $<i>S^*<l>$ with $S_{j,1}$ inserted after the trip in $<i>S^*<l>$ terminating at θ_{j_1} and $S_{j,2}$ inserted before the trip starting at $\sigma_{j_{\mu(j)}}$. Notice that within each set V'_{j_k} there are no forward links. The only forward links are through the gaps δ_{j_k} for $0\leq k\leq m(j)$-1, and $m(j)+1\leq k\leq \mu(j)$. Thus

$$AOv(<i>S_j^*<l>) = AOv(<i>S^*<l>) + 2\sum_{k=0}^{\mu(j)} \delta_{j_k} - 2 \max_{0\leq k\leq \mu(j)} \delta_{j_k}. \square$$

Denote the route with all the sets $V_{i+1}, \ldots, V_{l-1}$ inserted into $<i>S^*<l>$ using the method of lemma L3.10 as $<i>S^{**}<l>$.

Notice that for each j, the sets $V_{j_1}, \ldots, V_{j_{\mu(j)}}$ are inserted with $\mu(j)$ forward links. The forward link due to each V_{j_k} is either the distance $(bottom(V_{j_k}) - \theta_{j_k}) = \delta_{j_{k-1}}$ or

$(\sigma_{j_k} - top(V_{j_k})) = \delta_{j_k}$. Let $IC(V_{j_k})$ be the cost of inserting V_{j_k} as in L3.10. Then for any V_{j_k} such that $k \leq m(j)$, where $m(j)$ is the index of the maximum δ_{j_k}, V_{j_k} is inserted after θ_{j_k} at cost $IC(V_{j_k}) = 2\delta_{j_{k-1}}$. For V_{j_k} such that $k > m(j)$, V_{j_k} is inserted before σ_{j_k} at cost $IC(V_{j_k}) = 2\delta_{j_k}$. Using this notation,

$$AOv(<i>S^{**}<l>) = (bottom(V_i) - bottom(\Gamma)) + (top(\Gamma) - top(V_l)) + \sum_{k=i+1}^{l-1} IC(V_k).$$

For every combination of i and l such that $0 \leq i < l \leq v+1$, there is a route $<i>S^{**}<l>$ constructed using the methods of L3.8, L3.9, and L3.10. Let R^* be the one with least additional overhead. Thus

$$\begin{aligned} AOv(R^*) &= \min_{0 \leq i < l \leq v+1} \{AOv(<i>S^{**}<l>)\} \\ &= \min_{0 \leq i < l \leq v+1} \left\{ \sum_{k=i+1}^{l-1} IC(V_k) + (bottom(V_i) - bottom(\Gamma)) + (top(\Gamma) - top(V_l)) \right\}. \end{aligned}$$

R^* can be constructed directly without building all routes of the form $<i>S^{**}<l>$. To determine those sets that accumulate less additional overhead by prepending to S^* as opposed to inserting, it is sufficient to find i such that $(bottom(V_i) - bottom(\Gamma)) - \sum_{k=1}^{i} IC(V_k)$ is minimal. The procedure used to determine the optimal i is included in the next section where the complete algorithm for finding R^* is outlined. The idea is that the first index p such that $bottom(V_p) - bottom(\Gamma) < \sum_{k=1}^{p} IC(V_k)$ indicates that the sets $V_1, \ldots, V_p$ are less expensive to prepend than to insert. After this point, further savings are detected by searching for a new point $p' > p$ such that $(bottom(V_{p'}) - bottom(V_p)) < \sum_{k=p+1}^{p'} IC(V_k)$. Since the search is always for further savings, the largest p' found indicates those sets $V_1, \ldots, V_{p'}$ that should not be inserted.

Similarly, to determine those sets that accumulate less additional overhead through appending to S^* than through inserting, it is sufficient to determine l such that $(top(\Gamma) - top(V_l)) - \sum_{k=l}^{v} IC(V_k)$ is minimal.

If $l \leq i$ then, for the collection $V_l, \ldots, V_i$, prepending or appending both surpass inserting. Then $R^* = <i>S^*<i+1>$ for some i, and

$$\begin{aligned} AOv(R^*) &= (bottom(V_i) - bottom(\Gamma)) + (top(\Gamma) - top(V_{i+1})) \\ &= (top(\Gamma) - bottom(\Gamma)) - (top(V_{i+1}) - bottom(V_i)) \end{aligned}$$

But this is least when $(top(V_{i+1}) - bottom(V_i))$ is greatest. Therefore R^* can still be easily determined by finding i such that $(top(V_{i+1}) - bottom(V_i))$ is maximal.

3.4.3. Optimality of R^*

It remains to show that R^* is an optimal route through Γ.

Theorem T3.1

$$AOv(R^*) = \min\{AOv(R) \mid R \text{ is any route through } \Gamma\}.$$

proof:

Let $R_\alpha = (T_{\alpha_1}, \ldots, T_{\alpha_n})$ be any route through Γ. Let $b = src(T_{\alpha_1})$, and $e = dest(T_{\alpha_n})$. If $e < b$ then $AOv(R_\alpha) > AOv(<i>S^{**}<i+1>) \geq AOv(R^*)$. Therefore assume $e \geq b$. It is sufficient to show that $AOv(R_\alpha) \geq AOv(R^*)$. The points b, and e partition the set $V_1, \ldots, V_v$ into three parts: 1) $V_1, \ldots, V_q$, where $bottom(V_q) \leq b$, 2) $V_r, \ldots, V_v$ where $top(V_r) \geq e$, and 3) $V_{q+1}, \ldots, V_{r-1}$. Consider the sets in the third partition. If there are none then

$$\begin{aligned} AOv(R_\alpha) &\geq (b - bottom(\Gamma)) + (top(\Gamma) - e) \\ &\geq (bottom(V_q) - bottom(\Gamma)) + (top(\Gamma) - top(V_r)) \\ &= AOv(<q>S{**}<r>) \\ &\geq AOv(R^*) \end{aligned}$$

Otherwise R_α requires at least one forward link for each of the sets $V'_{j_k} \in \{V_{q+1}, \ldots, V_{r-1}\}$, since each $I(V'_{j_k}) \subset S_j$ for one and only one j. But the shortest possible forward link for each V'_{j_k} is either $(bottom(V'_{j_k}) - \theta_{j_k})$ or $(\sigma_{j_k} - top(V'_{j_k}))$. The route $<q>S^{**}<r>$ has the property that precisely the smallest possible forward links of these occur and no others. Therefore $AOv(R_\alpha) \geq AOv(<q>S^{**}<r>) \geq AOv(R^*)$. □

3.5. Statement of the Algorithm

The optimal route R^* is constructed by building up a linked list of the trips in the correct order. The basic structure of the algorithm is outlined but the detail of lower level routines is omitted. The bookkeeping necessary to execute these procedures without backtracking is not included.

Algorithm for 1Way del–1

```
Step 1:
order the trips yielding $\Gamma = (T_1, \ldots, T_n)$
such that $src(T_i) \leq src(T_{i+1})$.
```

Step 2:
# build S^*, the maximal subroute such that $AOv(S^*) = 0$ #

```
$subR \leftarrow T_1$          # put $T_1$ in the subroute #
Ex $\leftarrow \emptyset$           # Ex is the set of current extra trips #
For $j$ from 2 to $n$
do
    if $src(T_j) \leq dest(subR)$
    then
        # in same gapless set #
        if $dest(T_j) \geq dest(subR)$
        then
            # a higher staircase trip #
            add_to _stair($T_j$)
        else
            # not a staircase trip #
            if $Ex = \emptyset$
            then
                # $T_j$ becomes the current extra trip #
                start_new_Ex($T_j$)
            elif top($Ex$) $\geq$ $src(T_j)$
            then
                # $T_j$ goes into current extra set #
                add_to_Ex($T_j$)
            else
                # $T_j$ does not intersect Ex.
                therefore new Ex set #
                # deal with current Ex set #
                if $I(Ex) \cap link(j) \neq \emptyset$
                then
                    # Ex can be put into subR #
                    insert(Ex,j)
                else
                    # add Ex to list of V sets #
                    add_to_V(Ex)
                fi
                start_new_Ex($T_j$)
            fi
        fi
    else
        # new gapless set #
        # deal with Ex set as above and #
        add_to _stair($T_j$)
    fi
od
```

```
Step 3:
# Find those sets $V_1, \ldots, V_i$ that are better appended than inserted.
This step is specified in more detail than others,
as promised in the previous section. #

# initialization #
last_prepend_pt ← bottom($\Gamma$)      # bottom of last set prepended #
highest_prepend_set ← 0         # index of last set prepended #
total_IC ← 0                    # insert cost so far #
$i$ ← 1                           # index for current V-set #

while $i \leq v$                      # still V-sets left #
do
    currST ← stair($V_i$)           # stair containing $V_i$ #
    maxgap ← lowergap ($V_i$)       # gap below $V_i = \delta_{i-1}$ #
    last_low_insert ← $i$-1         #index of last set inserted from below#

    while $I(V_i) \subset$ currST
    do
        # decide whether to prepend or insert #
        # first find best insertion -- i.e. below or above #

        best_insert ← min {maxgap, uppergap($V_i$)}
        if (bottom($V_i$)-last_prepend_pt) $\leq$ (total IC + 2 best_insert)
        then
            # prepend $V_i$ #
            highest_prepend_set ← $i$
            maxgap ← uppergap($V_i$)
            last_prepend_pt ← bottom($V_i$)
            total_IC ← 0
        else
            # better so far to insert $V_i$ #
            if maxgap > uppergap($V_i$)
            then
                # $V_i$ best so far inserted before $\sigma(V_i)$ #
                insert_pt ($V_i$) ←"before $\sigma(V_i)$"
                total IC ← total IC + 2 uppergap($V_i$)
            else
                # $V_i$ and all lower sets in $S_j$ that are not
                prepended, are inserted after $\theta(V_k)$ since new maxgap #
                for k from last_low_insert+ 1 to $i$
                do
                    # change insertion point for $V_k$ #
                    insert_pt ($V_k$) ← "after $\theta(V_k)$"
                od
                last_low_insert ← $i$
                total_IC ← total IC + 2 maxgap
                maxgap ← uppergap ($V_i$)
            fi
            $i$ ← $i$+1 # ready to look at next $V_i$ #
        fi
    od
od
```

```
Step 4:
# Find those sets that are better appended than inserted.
This step has a structure mirroring Step 3. #

Step 5:
# If some sets are both below the highest prepend point and
above the lowest append point then find the largest gap in
the V-sets. #

if highest_prepend_set $\geq$ lowest_append_set
then
    highest_prepend_set $\leftarrow$ $i$ such that (top($V_{i+1}$) - bottom($V_i$))
        is maximum
    lowest_append_set $\leftarrow$ $i+1$
fi

Step 6:
# Build $R^*$ #

prepend ($V_1, \ldots, V_{highest\ prepend}$)
append ($V_{lowest\ append}, \ldots, V_v$)
for k from highest_prepend_set+1 to lowest_append_set-1
do  insert ($V_k$)
od
```

3.6. Time Complexity of the Algorithm

The steps outlined in the algorithm can be examined for timing requirements.

Step 1:

Since this is just a sort by sources, time required is $O(n \log n)$.

Step 2:

Each trips is tested once in the order returned from Step 1. With the appropriate bookkeeping each trip can be dealt with in constant time. Therefore this step requires $O(n)$.

Step 3:

Step 2 can pass all the required values for each V-set. The inner "while" loop increments i on each iteration. Since i is never decremented, the outer loop ensures there are no more than v passes through the inner loop all together. The only part of this loop that includes more iteration is the "for" loop. But each set can be examined here at most once. Therefore this step requires only $O(v)$ time.

Step 4:

As for Step 3, $O(v)$ is sufficient.

Step 5:

A single pass through each V-set record locates the largest $(top(V_{i+1}) - bottom(V_i))$, and therefore $O(v)$ time is used in this step.

Step 6:

Staircases need to be built and pointers rearranged as previously determined. If bookkeeping is thorough, this requires only $O(n)$ time at most.

Since $v < n$, steps 2 through 6 execute in $O(n)$. Thus ordering the trips is the most expensive part of the algorithm, which operates in time $O(n \log n)$.

The following lemma shows that $O(n \log n)$ is a lower bound on the time required to solve the *1Way del-1* problem, and thus the algorithm is of optimal order.

Lemma L3.11

$O(n \log n)$ is the optimal order for solving *1Way del-1*.

proof:

Let A be any algorithm that solves *1Way del-1*. Let $S = \{s_1, \ldots, s_n\}$ be a set of n integers. A can be used to sort the elements of S as follows. Construct a trip $T_i = (s_i, s_i + \frac{1}{2})$ for each element $s_i \in S$. Let I be an instance of *1Way del-1* consisting of the collection of these trips. Since the trips in I do not overlap, the algorithm A will produce the route $R = (T_{i_1}, \ldots, T_{i_n})$ such that $src(T_{i_j}) < src(T_{i_{j+1}})$ for every j. This yields an ordering of S such that $s_{i_1} < s_{i_2} < \cdots < s_{i_n}$. Thus A can be used to sort S. But sorting cannot be done in less than $O(n \log n)$ time. Hence algorithm A takes at least $O(n \log n)$ time. The given algorithm for *1Way del-1* achieves this lower bound and therefore is of optimal order. □

CHAPTER 4

THE ONE DIRECTIONAL, CAPACITY K $>$ 2, DELIVERY PROBLEM

Although the one directional delivery problem with capacity one can be solved in polynomial time, the problem is *NP*-complete for capacity greater than or equal to three. This is the content of this chapter. The problem for capacity two is so far unsolved and is discussed further in Chapter 5.

4.1. Definitions

A set of trips, Γ, has the same meaning here as in Chapter 3. Other definitions, with the exception of *route,* such as *src, dest, ls, gd, top,* and *bottom* are also used here as in Chapter 3. For capacity $k > 1$ it is no longer true that a route is made up of trips joined together by links between them. Let Γ be a set of n one-directional trips, and let α be an arrangement of the set $\{1, \ldots, 2n\}$. For arbitrary fixed capacity, k, a **route,** R_α, is an ordering, $R = r_{\alpha_1}, \ldots, r_{\alpha_{2n}}$, of the $2n$ points $\{s_1, \ldots, s_n, d_1, \ldots, d_n\}$ of sources and destinations of trips in Γ according to the arrangement α. In addition, it is necessary that

1. For every i, s_i precedes d_i and
2. For every j, the number of sources up to and including position j minus the number of destinations up to and including position j is less than or equal to k.

These two conditions ensure that packages are picked up before they are delivered, and that the car is never carrying more than k packages at one time. Notice that this definition, for the case $k = 1$, is equivalent to the definition of a route in Chapter 3.

4.2. A Lower Bound for Route Length

Because trips can be combined together when capacity is greater than one, the overhead as defined in Chapter 3 no longer has meaning. Still a lower bound is useful; but one is derived here for the total route length rather than for the overhead. The **length, *Len*,** of a route is $Len(R_\alpha) = \sum_{i=1}^{2n-1} | r_{\alpha_i} - r_{\alpha_{i+1}} |$.

The sources and destinations of trips in Γ partition the line segment from $bottom(\Gamma)$ to $top(\Gamma)$ into no more than $2n-1$ pieces. Let $p_1, p_2, \ldots, p_q$, $(q \leq 2n)$ be the points corresponding to sources and destinations in Γ, taken in order from least to greatest. Let $\rho(i)$ be the number of trips containing the line segment $\overline{p_i\, p_{i+1}}$. Since k packages at most can be carried at one time, and since all trips are forward, there must be at least $2\left\lceil \frac{\rho(i)}{k} \right\rceil - 1$ passes through this line segment to complete these deliveries. Therefore a distance of at least $(p_{i+1} - p_i) \cdot \left(2\left\lceil \frac{\rho(i)}{k} \right\rceil - 1 \right)$ must be travelled for each line segment. Summing these over successive segments yields the following lower bound.

Lemma L4.1

A lower bound on the length of a route R for the *1Way del-k* problem on a set of trips $\Gamma = \{T_1, \ldots, T_n\}$ partitioning the line segment with points $p_1, \ldots, p_q$ is $2\sum_{i=1}^{q-1} (p_{i+1} - p_i) \cdot \left\lceil \frac{\rho(i)}{k} \right\rceil - (p_q - p_1)$.

If there is a route that starts at the least source, ends at the greatest destination, always travels forward with a full car, and backward with an empty car, then that route passes through $\overline{p_i\, p_{i+1}}$ exactly $2\left\lceil \frac{\rho(i)}{k} \right\rceil - 1$ times, and achieves this lower bound. Also, since every interval $\overline{p_i\, p_{i+1}}$ must be travelled at least $2\left\lceil \frac{\rho(i)}{k} \right\rceil - 1$ times, if a route travels any interval more times than that, it cannot achieve the lower bound.

4.3. Problem Definition

For capacity $= 1$, the optimization version of the one directional delivery problem was of interest. To prove *NP*-completeness of the capacity ≥ 3 problem, the corresponding decision version is necessary. The capacity $= k$, one directional decision delivery problem is defined as:

Decision 1Way del–k

Instance: A set Γ of n trips, $\{T_1, \ldots, T_n\}$, such that $src(T_i) < dest(T_i)$ and a bound $B > 0$.

Question: Does there exist a route, R, through Γ that has length less than or equal to B?

4.4. 1Way del-k is NP-complete for k $\geq$3

The reduction is from the **Exact Cover by Three Sets,** *(X3C),* problem. This problem was proved *NP*-complete by Karp[1972]. *X3C* is defined by:

X3C

Instance: A finite set X with $|X| = 3q$ and a collection $\Pi = \{C_1, \ldots, C_m\}$ of three element subsets of X.

Question: Does Π contain an exact cover for X? That is, does there exist a subcollection $\Pi' \subseteq \Pi$ such that for every $x \in X$, $x \in C_l \in \Pi'$ for exactly one l?

Notice that Π' is an exact cover for X if and only if $\bigcup_{C_i \in \Pi'} C_i = X$ and $|\Pi'| = q$.

4.4.1. Proof of Completeness for Capacity Three

Theorem T4.1

1Way del-3 is *NP*-complete.

proof:

1Way del-k, and hence *1Way del-3,* are in *NP* since the general delivery problem is in *NP* and this is a subproblem.

To show that *1Way del-3* is complete, *X3C* is transformed to it. Let an arbitrary instance of *X3C* be specified by $X = \{x_1, \ldots, x_{3q}\}$ and $\Pi = \{C_1, \ldots, C_m\}$ such that each

$C_l = \{x_{l_1}, x_{l_2}, x_{l_3}\}$ where $x_{l_j} \in X$. To construct an instance of 1*Way del*–3 from this, a line segment $m + 3q + 1$ units long is required, and points labelled at unit distances. One endpoint and the following $m-1$ points are consecutively labelled $C_1, C_2, \ldots, C_m$. The following $3q$ points are labelled $x_1, x_2, \ldots, x_{3q}$. The second last point is labelled *out* and the last point *in*. See Figure 4.1. This line segment is assumed directed from C_1 to *in*, and trips are embedded in this line segment always in this direction. The set Γ of constructed trips consists of $6m$ members, which can be partitioned into 3 sets.

The first set, Γ^1, only depends on the size of X. For every i from 1 to $3q$, construct the trip $T_i^1 = (x_i, in)$. Let $\Gamma^1 = \{T_1^1, \ldots, T_{3q}^1\}$.

The second set, Γ^2, depends on the number of occurrences of each x_i in the collection Π. Let $\eta(x_i)$ be the number of times that x_i appears in the collection Π. For each i from 1 to $3q$, construct $\eta(x_i)-1$ identical trips, $T_{i,j}^2 = (x_i, out)$; $1 \leq j \leq \eta(x_i)-1$. Let $\Gamma^2 = \bigcup_{\substack{1 \leq i \leq 3q \\ 1 \leq j \leq \eta(x_i)-1}} \{T_{i,j}^2\}$.

The set Γ^3 is determined by the contents of the sets C_l in Π. For each $C_l \in \Pi$ construct the three trips $T_{l,j}^3 = (C_l, x_{l_j})$ for $j = 1,2,3$. Let $\Gamma^3 = \bigcup_{\substack{1 \leq l \leq m \\ 1 \leq j \leq 3}} \{T_{l,j}^3\}$. Now set $\Gamma = \Gamma^1 \bigcup \Gamma^2 \bigcup \Gamma^3$, and let:

$$\begin{aligned} B &= 2\sum_{l=1}^{m}(out - C_l) + 2q(in - out) - (in - C_1) \\ &= 2m \cdot 3q + 2\sum_{l=1}^{m} l + 2q - (m + 3q + 1) \\ &= 6mq + m^2 - q - 1. \end{aligned}$$

The complete construction is illustrated in Figure 4.1.

Note that $|\Gamma^1| = 3q$, and $|\Gamma^3| = 3m$, and that each trip in these sets can be constructed in constant time. Since there are m 3-element subsets containing elements of X, the sum of the number of occurrences of all x_i in $\bigcup C_l$ is $3m$. Therefore, $\sum_{i=1}^{3q}(\eta(x_i)-1) = 3(m-q) = |\Gamma^2|$. These trips can also each be constructed in constant time. Determining $\eta(x_i)$ for all i requires a single

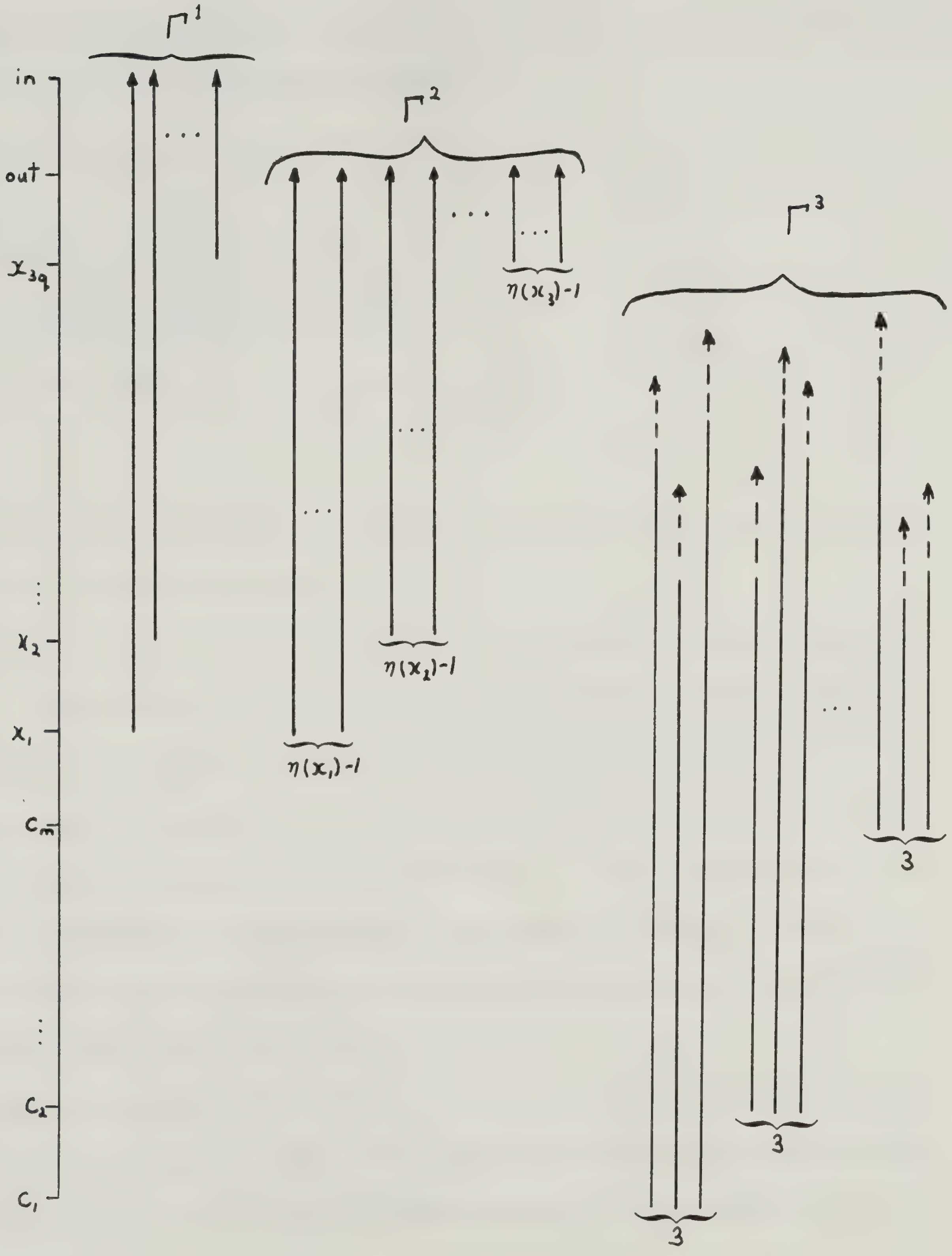


Figure 4.1 1Way del-3 reduces to X3C

pass over the sets in Π, and therefore takes time proportional to m. So $|\Gamma| = 3q + 3m + 3(m-q) = 6m$, and the construction of this instance of *1Way del*–3 from an instance of *X3C* is polynomial in m. It remains to show that Π contains an exact cover of X if and only if Γ has a route of length less than or equal to B.

By lemma L4.1, a lower bound on the length of a route for Γ is

$$2\sum_{i=1}^{m+3q+1}(p_{i+1}-p_i)\cdot\left\lceil\frac{\rho(i)}{3}\right\rceil-(in-C_1)$$

$$=2\sum_{i=1}^{m}1\cdot\left\lceil\frac{3i}{3}\right\rceil+2\sum_{i=1}^{3q}1\cdot\left\lceil\frac{3m}{3}\right\rceil+1\cdot2\left\lceil\frac{3q}{3}\right\rceil-(m+3q+1)$$

$$=2\left(\frac{m(m+1)}{2}\right)+3q\cdot2m+2q-(m+3q+1)$$

$$=6mq+m^2-q-1$$

But this is the bound B. Therefore it is sufficient to show that Π contains an exact cover if and only if Γ has a route of length exactly B.

Suppose that $\Pi' \subset \Pi$ is an exact cover of X. Let a route R for Γ proceed as follows. R starts at C_1 and picks up all 3 trips $T^3_{1,j}$. If $C_1 = \{x_{1_1}, x_{1_2}, x_{1_3}\} \in \Pi'$ then as the car passes each destination of $T^3_{1,j}$, it completes that trip and immediately picks up the trip (x_{1_j}, in) in Γ^1. These trips are completed by travelling to in. Otherwise, if $C_1 \notin \Pi'$, then, as the car passes each destination of $T^3_{1,j}$, it immediately picks up one of the trips (x_{1_j}, out) in Γ^2. These trips are all completed by travelling to out. In either case, the car has started at C_1 and travelled directly to in or out always maintaining a full load without backtracking. If the car ends at in after this first forward pass, it travels empty from in back to out.

For each remaining C_l that is not in Π', the car now travels from out to C_l and back to out. From out to C_l the car is empty. At C_l it picks up all 3 trips $T^3_{l,j}$, and changes direction heading for out. As it passes each of its current destinations, it picks up the trip $(x_{l_j}, out) \in \Gamma^2$ and proceeds to out.

Now for each remaining $C_k \in \Pi'$, the car moves directly back to C_k and then forward to *in*. Travelling backward to C_k, the car is empty. At each C_k it picks up the three trips $T^3_{k,j}$ and changes direction. As it passes each of its current destinations, it immediately picks up the corresponding trips $(x_{k_j}, in) \in \Gamma^1$ and proceeds to *in*.

Since this route starts at the least source and ends at the greatest destination, travels forward with a full car, and backward with an empty car, it must achieve the lower bound. Thus the route has length B.[1] Therefore, if Π contains an exact cover then there is a route for Γ with length less than or equal to B.

Conversely, suppose there is a route, R, for Γ with length B. Since B is the lower bound, R travels through each interval $\overline{p_i\, p_{i+1}}$, exactly $2\left\lceil \frac{\rho(i)}{3} \right\rceil - 1$ times, and therefore forward through each interval at most $\left\lceil \frac{\rho(i)}{3} \right\rceil$ times. But by the construction of Γ, each interval $\overline{p_i\, p_{i+1}}$ has a multiple of 3 trips passing through it. Each interval $\overline{C_l\, C_{l+1}}$ is contained by $3l$ trips; each interval from C_m to *out* is contained by $3m$ trips; and the interval $\overline{out\ in}$ is contained by $3q$ trips. Therefore, when R travels forward through an interval, it must have a full car. Also, for a trip T and any interval $\overline{p_i\, p_{i+1}} \subset T$, the package for T must be carried through that interval only one time. Otherwise all $\rho(i)$ trips in $\overline{p_i\, p_{i+1}}$ cannot be completed in $\left\lceil \frac{\rho(i)}{3} \right\rceil$ forward passes. This implies that when R travels backward, it must have an empty car.

Now consider the $3q$ trips containing $\overline{out\ in}$. R must complete them in q forward passes, each time executing 3 deliveries. On any one pass through $\overline{out\ in}$, let T_u, T_v, T_w be the trips being executed. These trips have unique sources, say x_u, x_v, x_w since all trips to *in* have unique sources. But then the three trips completed by R just prior to T_u, T_v, and T_w must have terminated at x_u, x_v, and x_w, since otherwise R would have to travel forward without carrying full capacity or backward without being empty. Let these trips be T'_u, T'_v, and T'_w and their

[1] This can also be seen by calculating the length of the described route directly.

sources be C_u, C_v, and C_w in $\{C_1, \ldots, C_m\}$. Since no trips terminate in this set, R must have been travelling backward before these trips. Hence $C_u = C_v = C_w$ since otherwise again R would be travelling forward not full or backward not empty. Similarly the remaining $q-1$ passes through $\overline{out\ in}$ isolate a unique point $C_k \in \{C_1, \ldots, C_m\}$. The sets in Π corresponding to these points form a cover for X of size q, which must therefore be an exact cover. Thus if there is a route R for Γ of length less than or equal to B, then Π contains an exact cover for X. □

4.4.2. Proof of Completeness for Capacity Greater Than 3

It remains to show that *1Way del-k* for any $k > 3$ is also *NP*-complete. This can be done by supplementing the previous construction with additional trips.

Corollary C4.1

1Way del-k, for any $k > 3$, is *NP*-complete.

proof:

1Way del-k is clearly in *NP* since a non-deterministic algorithm need only guess a feasible arrangement of the $2n$ points consisting of sources and destinations of trips and check that the corresponding length is less than or equal to B.

E3C is again used for the transformation. Let Γ be the set of trips constructed from an instance of *E3C* in the previous proof. For each l, $1 \leq l \leq m$, define $k-3$ identical trips $T'_{l,j} = (C_l, out)$ for $1 \leq j \leq k-3$. Also define $(k-3)q$ identical trips $T''_i = (out, in)$ for $1 \leq i \leq (k-3)q$. Let Γ_k^* be the set of trips Γ supplemented with these extra $(k-3)m + (k-3)q$ trips defined above. Set the bound B to be the same as defined in theorem T4.1. Then it is straightforward to see that there is an exact cover for X if and only if there is a k-capacity route through Γ_k^* of length B. The extra trips enforce a full car for all forward motion, and prevent simultaneous completion of trips starting at more than one point C_l. □

CHAPTER 5

CONCLUSIONS AND FURTHER RESEARCH

5.1. Observations Concerning the Delivery Problem

The question of why a problem is *NP*-complete, could have direct bearing on whether $P = NP$. Therefore the answer will typically be difficult. However, there is one part of the 1*Way del–k* problem for a constant k that can be isolated as being polynomially solvable, thus indicating that this part alone is not responsible for the difficulty of the problem. Suppose some oracle declared the optimal order in which to pick up the sources of a given collection of trips. Because of the two restrictions on feasible routes, 1) sources precede corresponding destinations, and 2) no more than k deliveries can be current, there are well defined limits on the intermediate insertion of destinations into the route. A dynamic programming approach requiring only polynomial time can be used to place the destinations optimally. This is shown for capacity $k=2$ in the section of this chapter entitled "Open Problems". In this case the insertion of destinations has complexity $O(n^2)$. The generalization to arbitrary fixed capacity k results in an algorithm with complexity $O(n^{k+1})$. Similarly, the sources can be inserted optimally into a fixed ordering of the destinations in polynomial time. It appears, therefore, that the problem's difficulty is due to the way that sources and destinations interact and cannot be attributed to either part individually.

Another observation involves the complicated solution to 1*Way del–k* presented in Chapter 3. A lot of apparent complexity is due to seeking an optimal route rather than an optimal tour. If just a tour had been required, the solution would have been much simplier, because the extra details to accommodate special properties of starting points and ending points would disappear. The solution becomes one of deriving the lower bound $\sum_{i=i}^{n} d_i - \sum_{i=1}^{n} s_i + 2\sum_{i=1}^{t} g_i$ for the overhead of any tour through Γ. This is just a corollary of lemmas L3.2, and L3.4. Then it is easy to show

that this lower bound is always attainable using the same method as used to derive the *IUB* in Chapter 3. This type of situation is not particularly rare. Frequently such a lack of symmetry hides a lot of inherent simplicity.

5.2. Corollaries to the Completeness Result

The *NP*-completeness of the two directional k capacity, delivery problem, *delivery*$(1D, 1W, kcap)$ follows from the completeness of $1Way\ del{-}k$, since the one way problem is a restricted version of the more general two way problem. Alternatively, this result can be proven directly using a modification of the reduction constructed in Chapter 4. For capacity 3, it is sufficient to construct $3q$ return trips from *in* to *out* and 3 trips from *out* to each of the C_i's except C_1. Then there is an exact cover if and only if there is a route such that the car is always full. The extension to capacity k simply mimics the extension for the one way case.

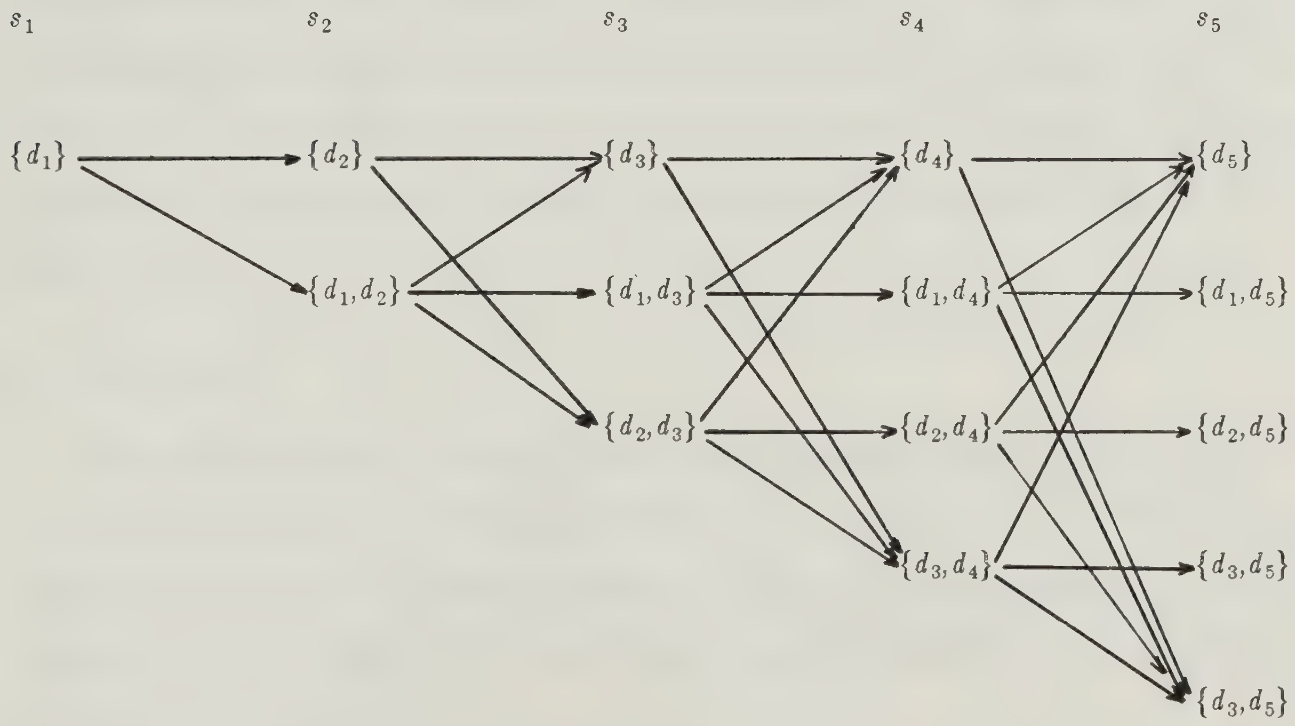
Another variation, called the **direct delivery problem**, requires that once a package is picked up at s_i, the car must travel directly to d_i without any backtracking. It is allowed to pick up further packages before delivering T_i only if they are *en route* to d_i. Even though this is a further restriction of $1Way\ del{-}k$, the direct delivery problem remains *NP*-complete. In fact the proof in Chapter 4 that $E3C$ reduces to $1Way\ del{-}3$, actually constructs an instance of the direct delivery problem, thus proving the claim.

5.3. Open Problems

The problems for capacity equal to 2, $1Way\ del{-}2$ and the two way version, *delivery*$(1D, 1W, 2cap)$, are unsolved. A completely different approach seems necessary in order to determine the status of these problems. The usefulness of $E3C$ for attacking this problem appears exhausted at capacity equal to 3.

However if the optimal ordering of the sources is known, then there is an algorithm that inserts the destinations optimally in $O(n^2)$ time. Let $S = (s_1, \ldots, s_n)$ be the oracle's ordering of sources for the optimal route. Construct a multi-stage graph with n stages. Each node in the

j^{th} column (stage) represents a possible configuration in the car immediately after picking up the source s_j. Thus there is one node for every possible set of undelivered destinations. At the j^{th} stage each of these sets contains destination d_j. A node may also contain 1 other undelivered destinations selected from the $j-1$ trips already picked up. Therefore the j^{th} stage contains $1 + \binom{j-1}{1} = j$ nodes. Figure 5.1 illustrates the multi-stage graph for $n = 5$ when capacity $k = 2$. Nodes may have a size of 1 (just d_j) or 2 (a full car). Nodes of size 2 have three successors since either 1 or 2 different destinations are visited before going to the next source. Nodes of size 1 have two successors since the current destination can be either visited or not before going to the next source. Therefore the number of edges going from nodes at stage j to nodes at stage $j+1$ is $2 + 3(j-1) = 3j - 1$.



$$\begin{tikzcd}
s_1 & s_2 & s_3 & s_4 & s_5 \\
\{d_1\} \arrow[r] \arrow[dr] & \{d_2\} \arrow[r] \arrow[ddr] & \{d_3\} \arrow[r] \arrow[dddr] & \{d_4\} \arrow[r] \arrow[ddddr] & \{d_5\} \\
 & \{d_1,d_2\} \arrow[ur] \arrow[r] \arrow[dr] & \{d_1,d_3\} \arrow[ur] \arrow[r] \arrow[ddr] & \{d_1,d_4\} \arrow[ur] \arrow[r] \arrow[dddr] & \{d_1,d_5\} \\
 & & \{d_2,d_3\} \arrow[uur] \arrow[r] \arrow[dr] & \{d_2,d_4\} \arrow[uur] \arrow[r] \arrow[ddr] & \{d_2,d_5\} \\
 & & & \{d_3,d_4\} \arrow[uuur] \arrow[r] \arrow[dr] & \{d_3,d_5\} \\
 & & & & \{d_3,d_5\}
\end{tikzcd}$$

Figure 5.1 Multi-stage graph for 5 trips

Since an edge between a node at stage j and a node at stage $j+1$ represents the delivery of 0, 1, or 2 trips, the optimal ordering for these deliveries can be found in constant time, say K. Let the cost of calculating all edges between stage i and stage j be denoted $C_{i,j}$. Then the cost of calculating all edges between stage j and stage $j+1$ is bounded by $C_{j,j+1} \leq K \cdot (3j - 1)$. Summing over all stages yields $C_{1,n} \leq \sum_{j=1}^{n-1} K \cdot (3j - 1) = O(n^2)$. A final stage of the multi-stage graph is required to represent the delivery of the destinations remaining after s_n is picked up. This cost is bounded by $O(n)$. Thus the destinations can be inserted optimally into the ordering of the sources in time proportional to n^2.

Other variations could be considered. If the packages on the car waiting for delivery form a stack or a queue the problem changes significantly. It is not known if this restriction is sufficient to make the problem a member of P. As previously noted, the variation where the car travels on a simple closed curve, has been studied in the one directional case. It seems likely that the corresponding two directional case can be solved by combining some results from both the two directional line segment version and the one directional circular version. In any case, except when a particular application warrants it, pursuing numerous slight variations does not appear to be a useful endeavour.

5.4. Applications

The problem *delivery*(1*D*,1*W*,1*cap*) originated as a manufacturing industry problem which was erroneously thought to be *NP*-complete. Efficient delivery systems are one application of the work presented in this thesis. However the primary results of this thesis, determining a set of restrictions that can be applied to a problem without significantly simplifying it, are at present essentially of theoretical interest.

REFERENCES

Chang, Robin [1976], "An overview of the Frito-Lay system," unpublished manuscript.

Cook, S. A. [1971], "The complexity of theorem proving procedures," *Proceedings of the Third Annual ACM Symposium on Theory of Computing,* Association for Computing Machinery, New York, 151-158.

Garey, M. R., D. S. Johnson [1979], *Computers and Intractability A Guide to the Theory of NP-Completeness,* W. H. Freeman and Company, San Francisco.

Gardner, Martin [1973], "Up and down elevator games and Piet Hein's mechanical puzzles," in Mathematical Games, *Scientific American,* February 1973,106-109.

Gilmore, P. C., R. E. Gomory [1964], "Sequencing a one state-variable machine: a solvable case of the traveling salesman problem," *Operations Research* 12, 655-679.

Karp, R. M. [1972], "Reducibility among combinatorial problems," *Complexity of Computer Computations,* R. E. Miller and J. W. Thatcher (eds.), Plenum Press, New York, 85-103.

Knuth, D. E. [1973], *The Art of Computer Programming, Volume 3: Sorting and Searching,* Addison-Wesley, Menlo Park, California.

Liu, P. C. [1976], *Analysis of Non-Planar Graphs,* Ph.D. dissertation, Stevens Institute of Technology, Castle Point, New Jersey.

Papadimitriou, C. H. [1977], "The Euclidean traveling salesman problem is NP-complete," *Theoretical Computer Science* 4, 237-244.

Wong, C. K., C. L. Liu, J. Apter [1973], "A drum scheduling algorithm," *Proceedings of the l, Fachtagung uber Automatentheorie und Formale Sprachen,* July, 1973, Bonn, 267-275.